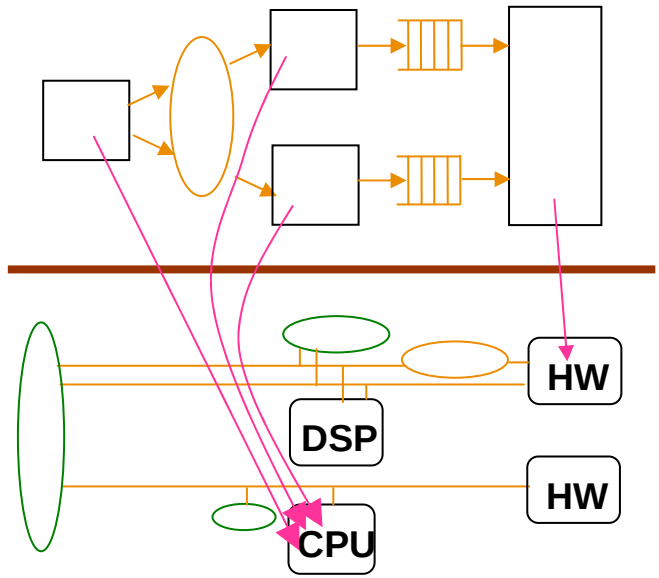
Mapping and architecture exploration

• Configure the resources, e.g. the size of an internal memory, width of a bus.

• Map the processes to the resources.

• “Compile” the processes in terms of the “services” provided by the mapped resources:



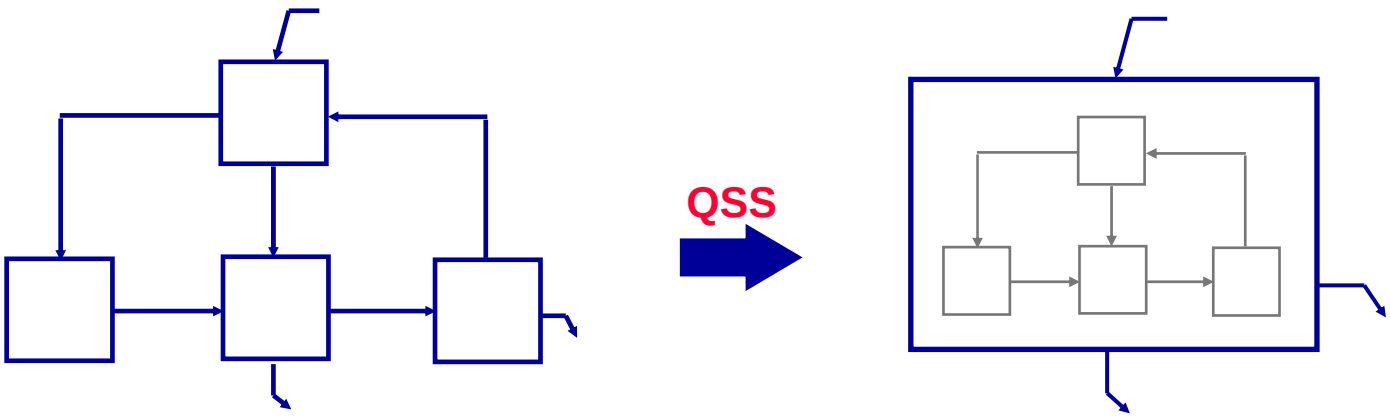
Effective scheduling of process operations mapped to a CPU is a key issue:

- reduce the context-switching between tasks for efficient execution
- increase data coherency among processes for efficient memory usage

Scheduling Classification

- **Dynamic scheduling**
  - Make all scheduling decisions at run-time
  - Context switch overhead
- **Static scheduling**
  - Make all scheduling decisions at compile-time
  - Reduce context switch overhead
  - Restricted to specification without data-dependent controls (e.g. if-then-else)
- **Quasi-static scheduling**
  - Allow specification to have data-dependent controls
  - Perform static scheduling as much as possible
  - Leave data-dependent choices resolved at run-time

Quasi-static scheduling

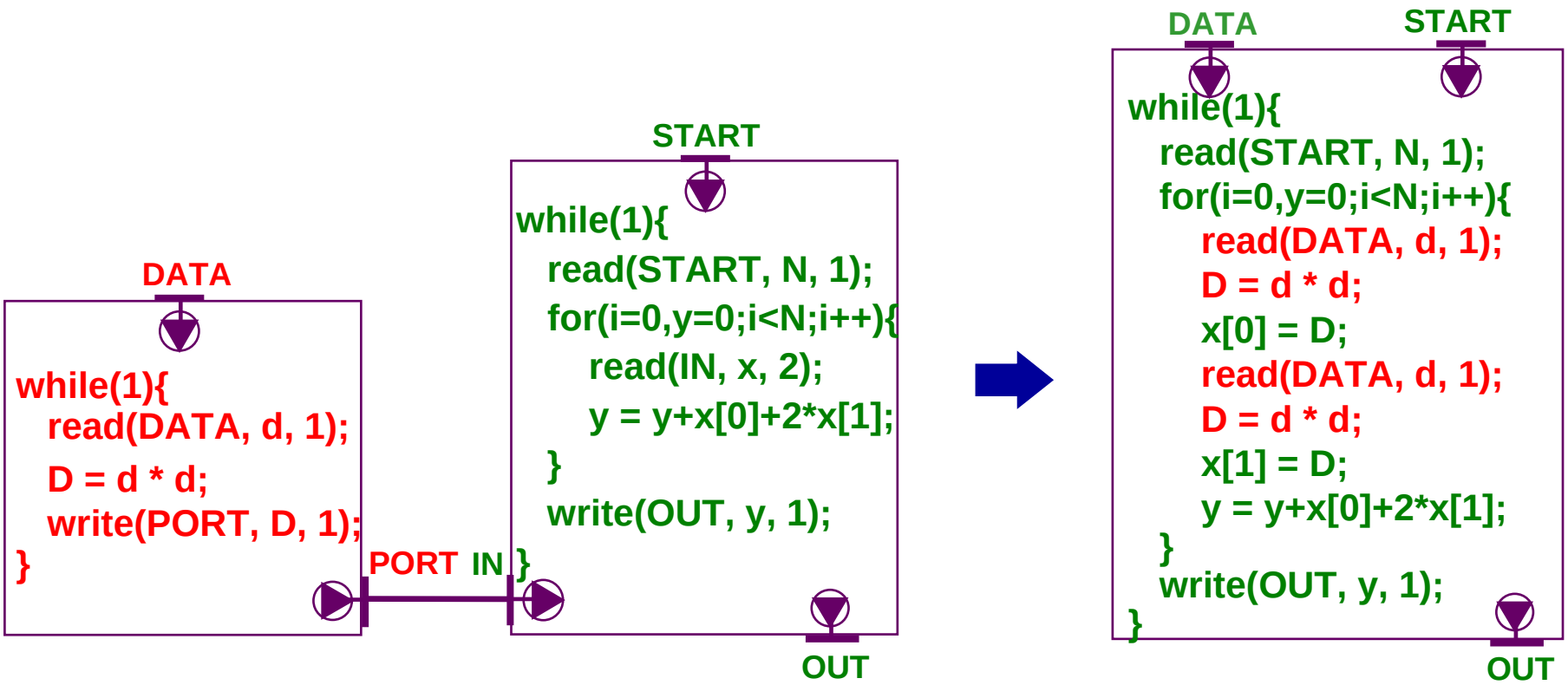


Sequentialize concurrent operations as much as possible.

➡ A better starting point for code generation technologies:

- Straight-line code across function blocks
- Bounded memory usage during the code execution

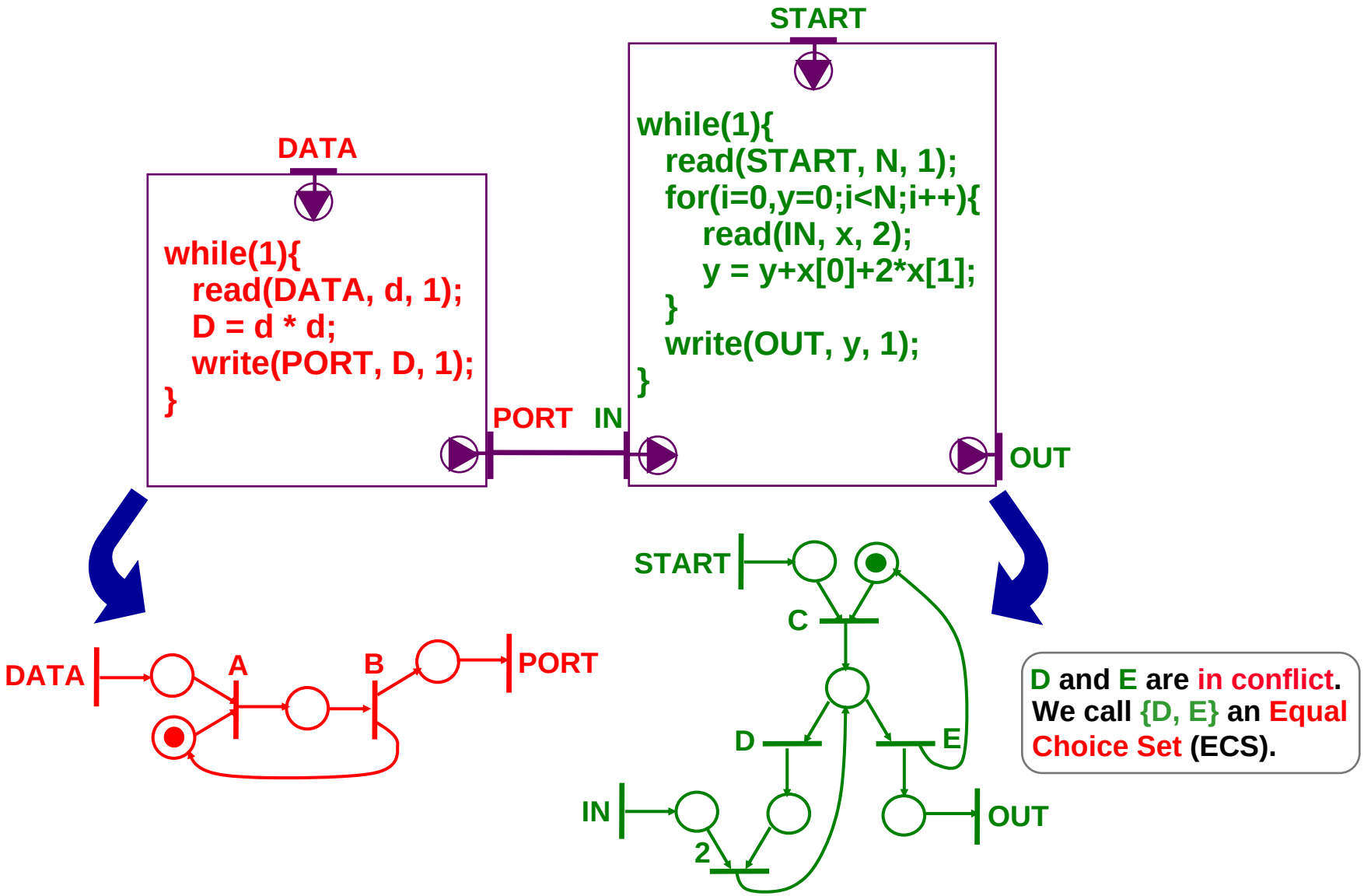
Scheduling concurrent programs



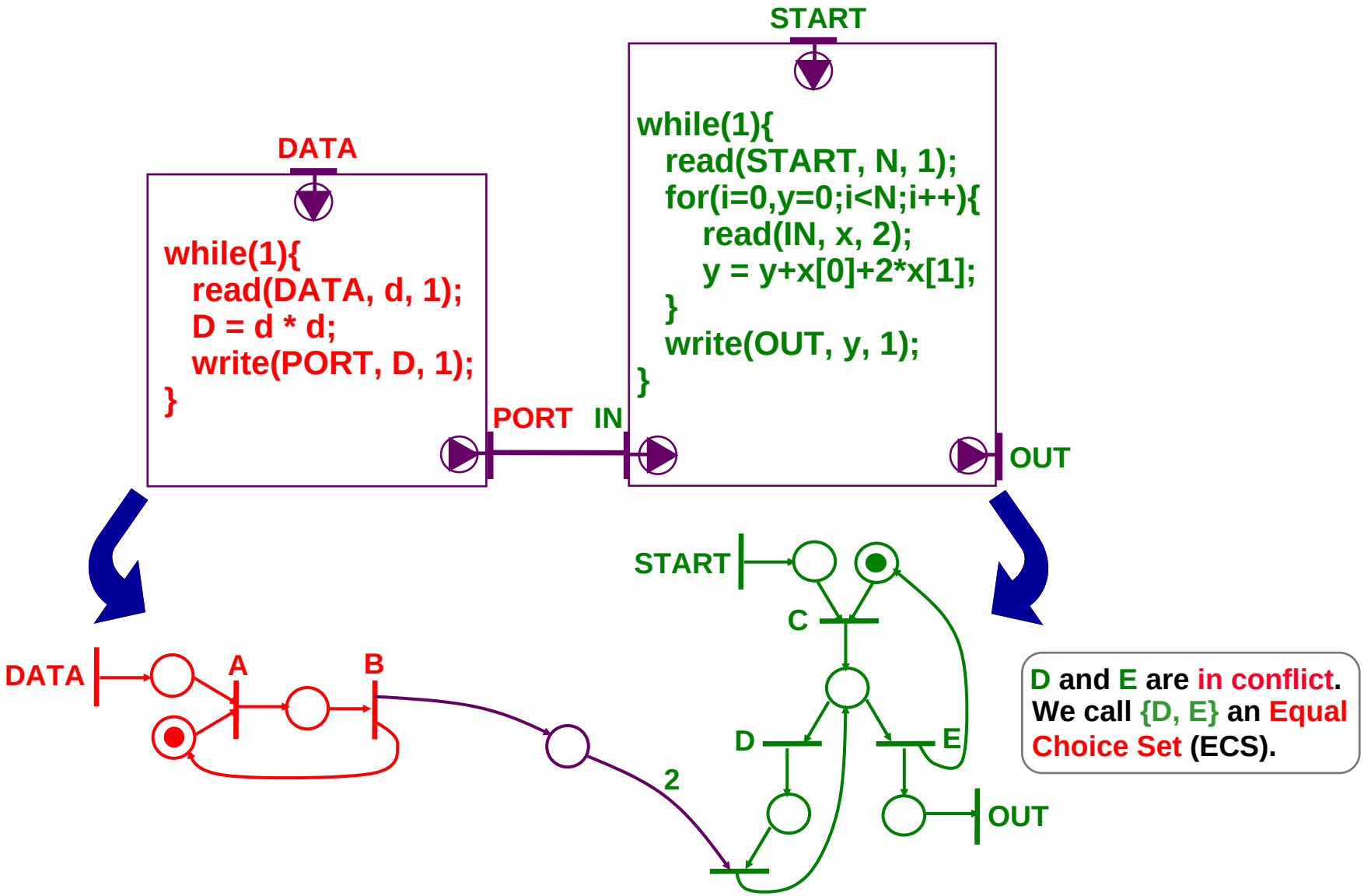
Outline

- The problem and an approach
  - Petri Net: a model for capturing the behavior of a program
  - Definition of a schedule
  - An algorithm
- An application to MPEG2 decoder
  - The effectiveness
  - The outstanding problem: False Path Problem
- Approaches for the False Path Problem
  - Cong's observation
  - Future directions

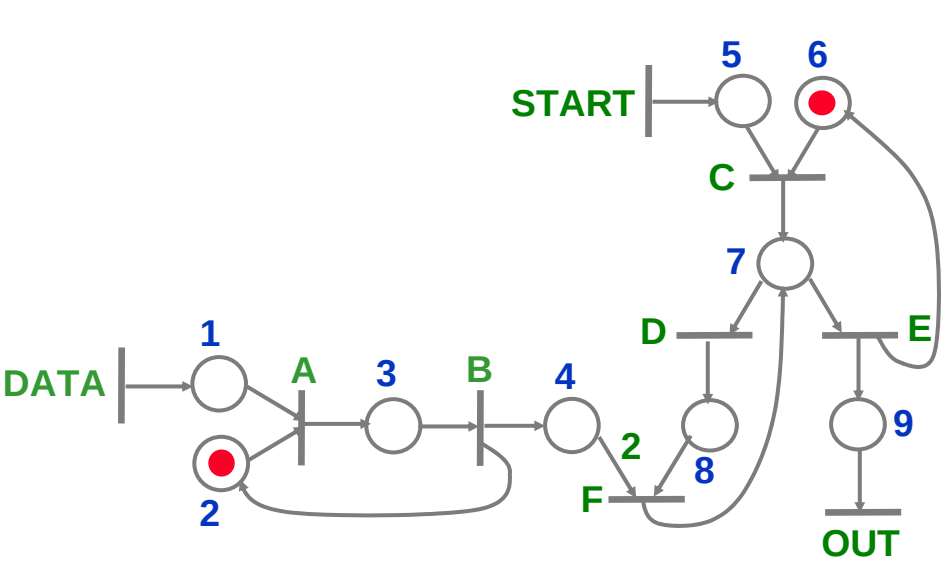
Scheduling concurrent programs

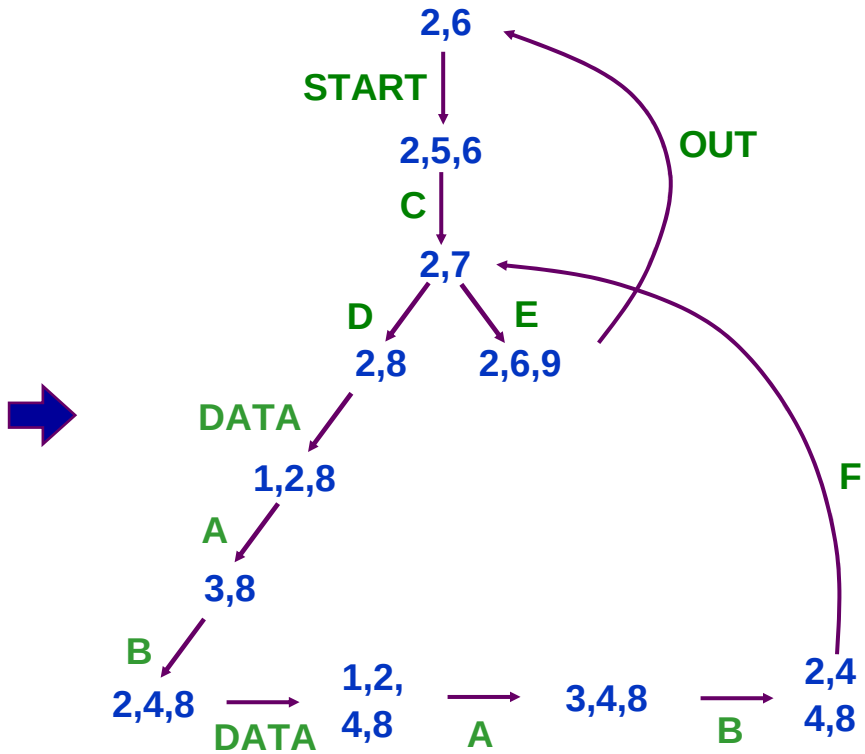


Scheduling concurrent programs



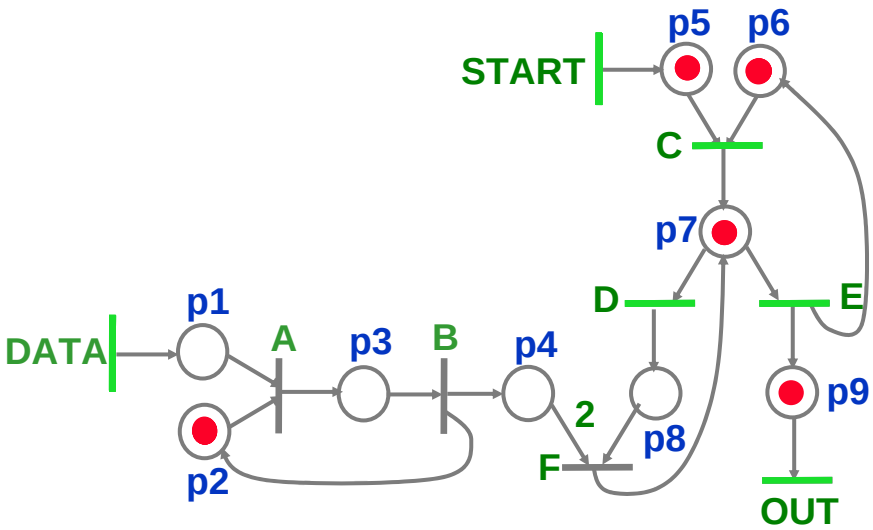
Scheduling concurrent programs

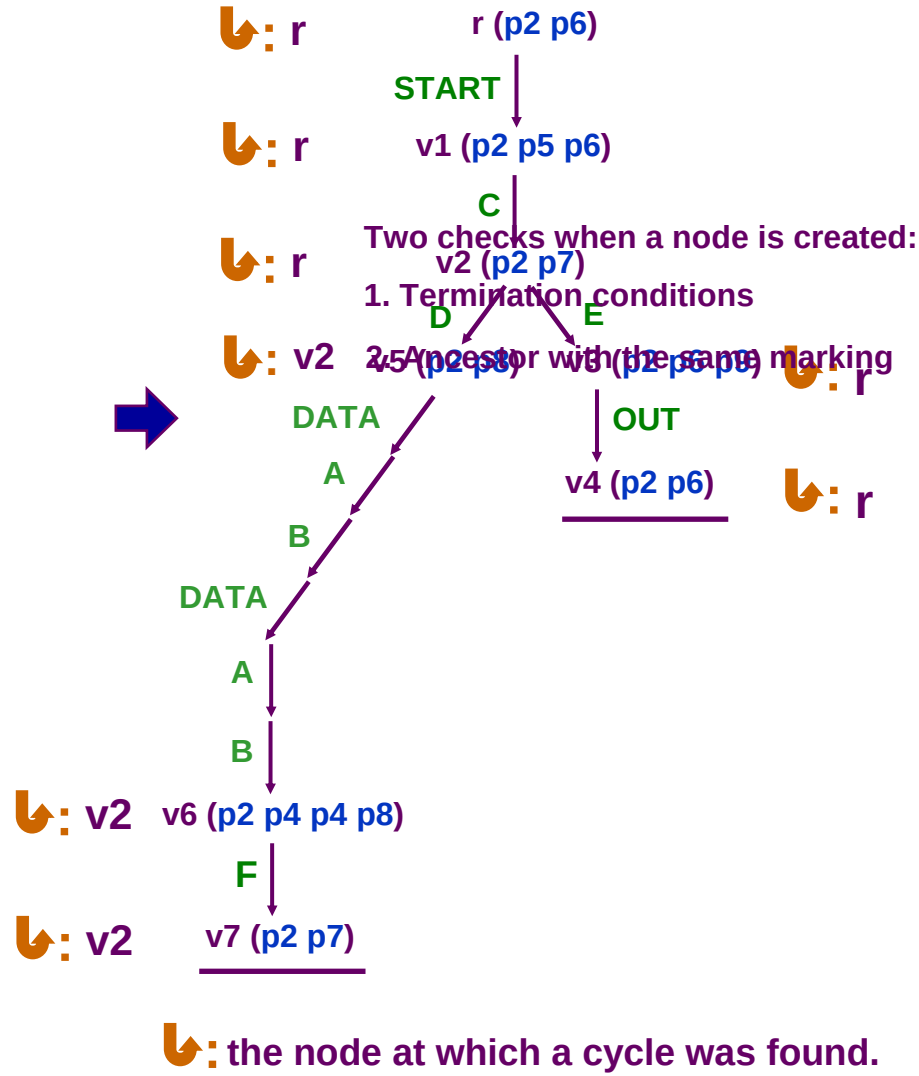




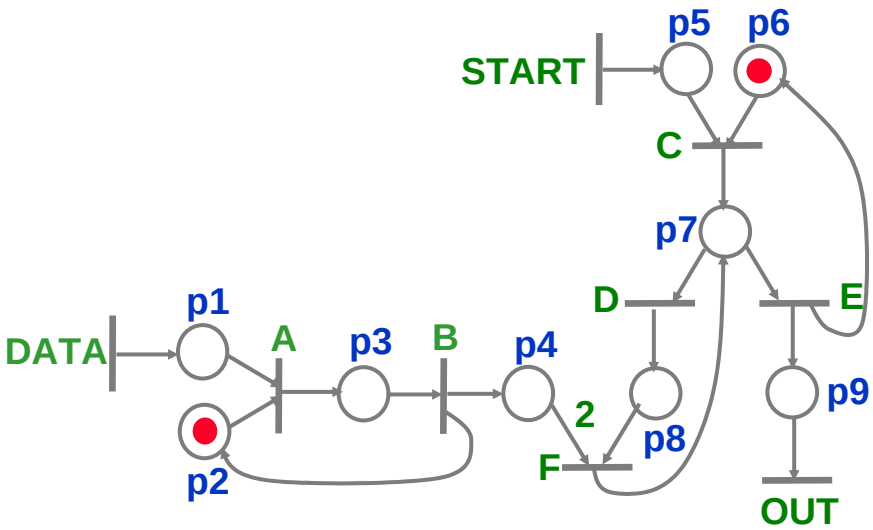
- One node r associated with the initial marking.
- All and only transitions of an enabled ECS from each node.
- A path to the node r from each node.

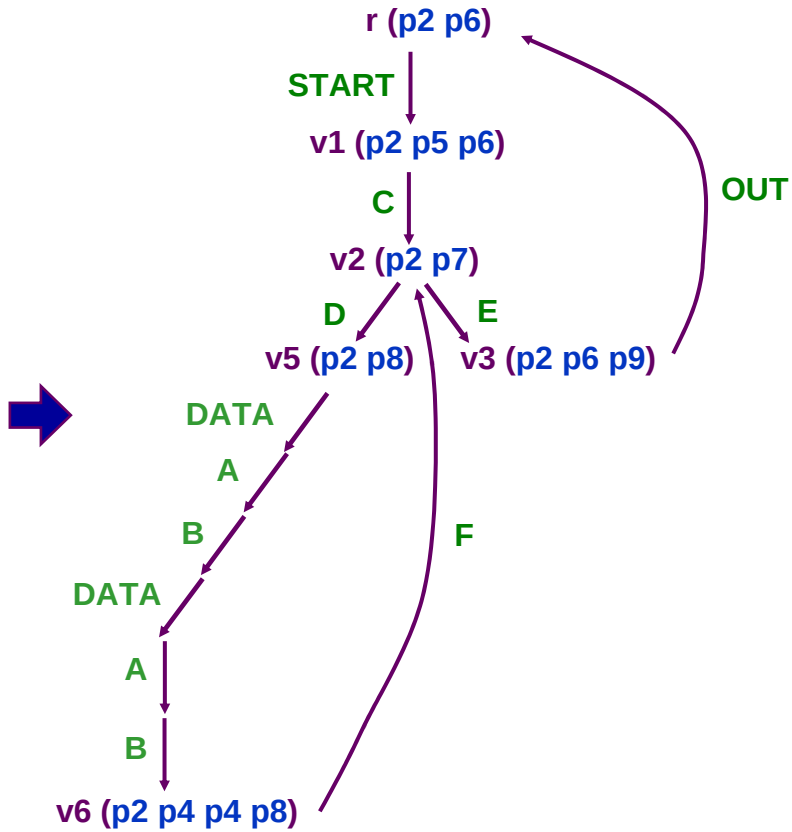
Finding a schedule on the Petri net



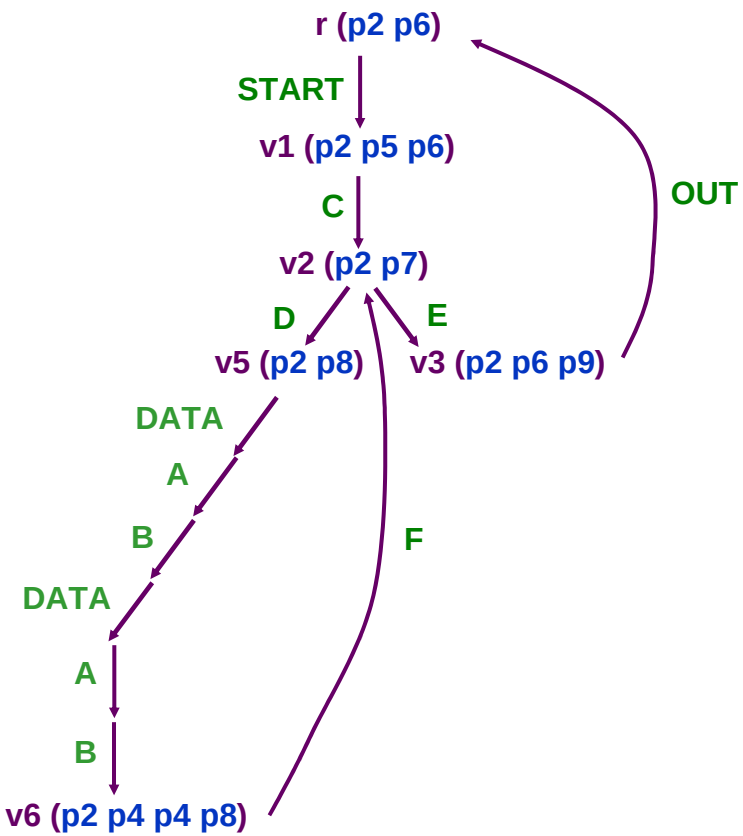


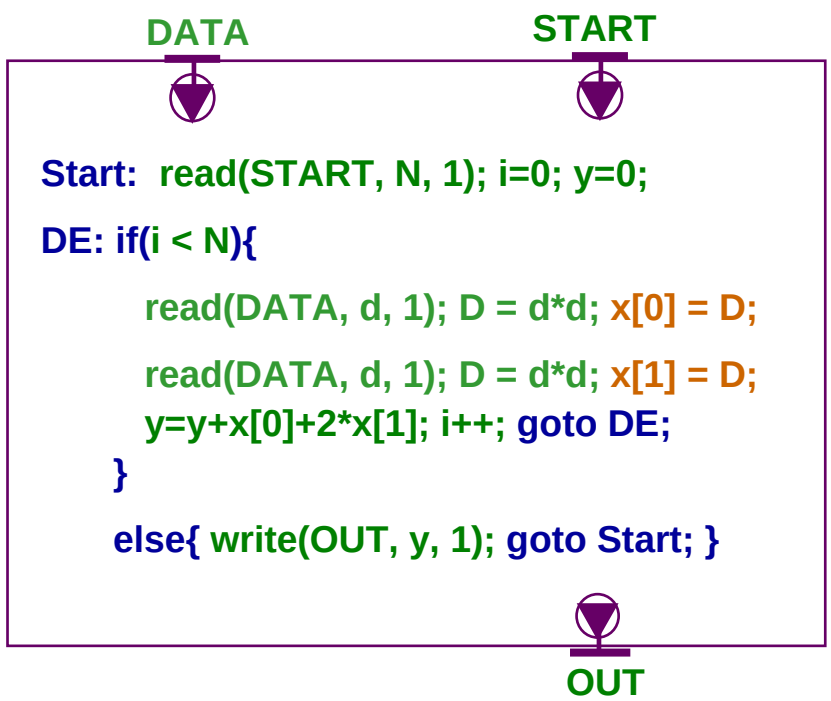
Finding a schedule on the Petri net





Generating code from a schedule





Properties of the algorithm

- **Claim1:**

  If the algorithm terminates successfully, a schedule is obtained.

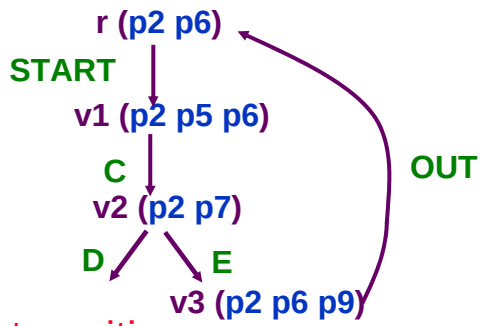
  - The schedule provides an upper bound on memories required for communicating data.

- **Claim2:**

  If the algorithm does NOT terminate successfully, no schedule exists under given termination conditions.

Improving efficiency

- Which transition to choose at each node?



  - Find sequences of transitions to create cycles.

    T-invariants: a basis of the linear system **A x = 0**

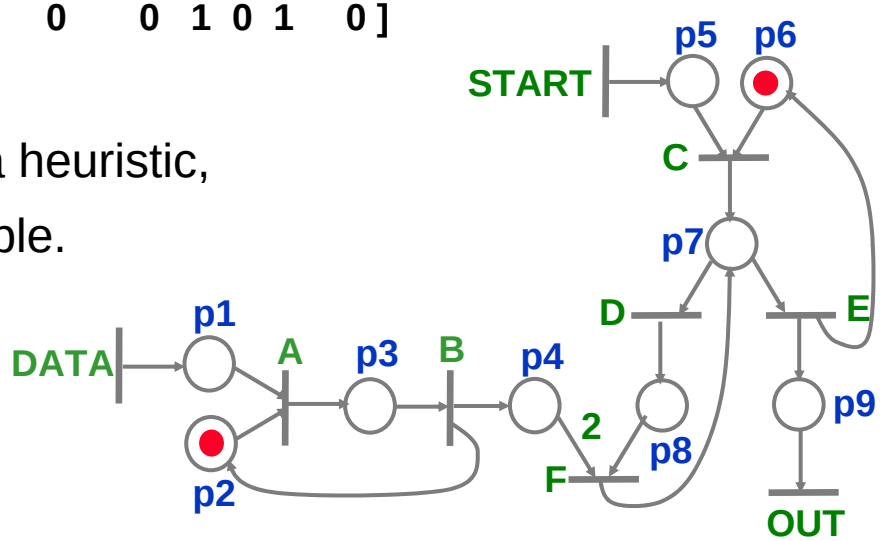
    A[i, j]: # of tokens produced to the i-th place by the j-th transition.

T-invariants:

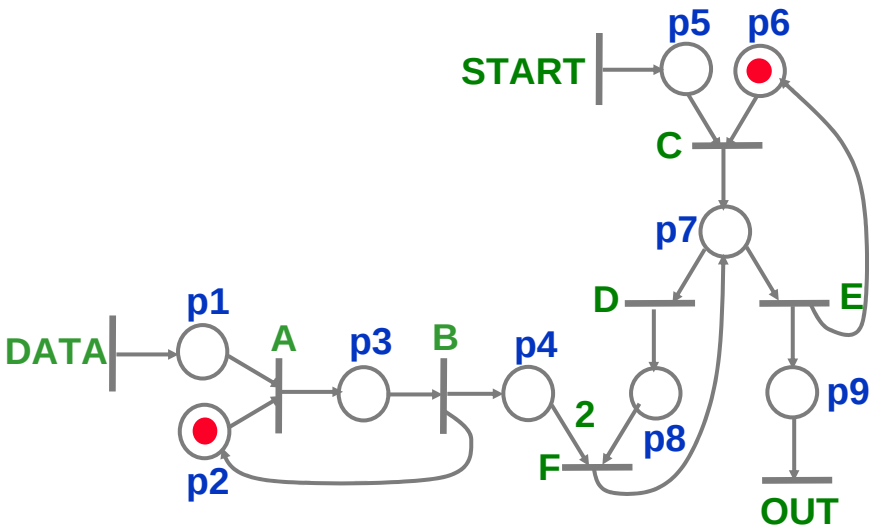
DATA	A	B	START	C	D	E	F	OUT
[0	0	0	1	1	0	1	0	1]
[2	2	2	0	0	1	0	1	0]

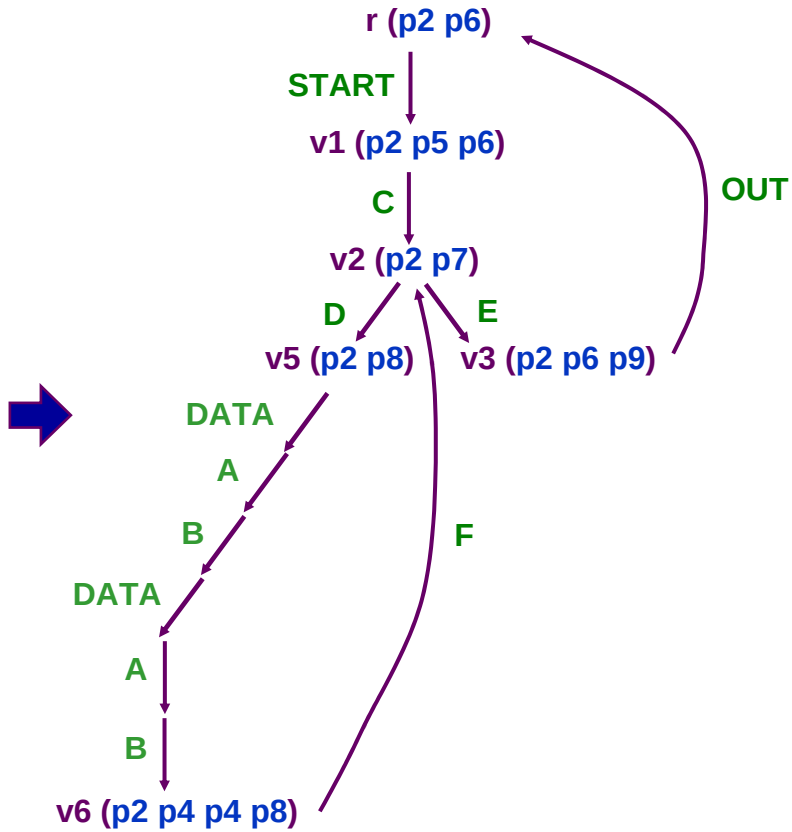
  - Choose a T-invariant using a heuristic,

    and use it as much as possible.



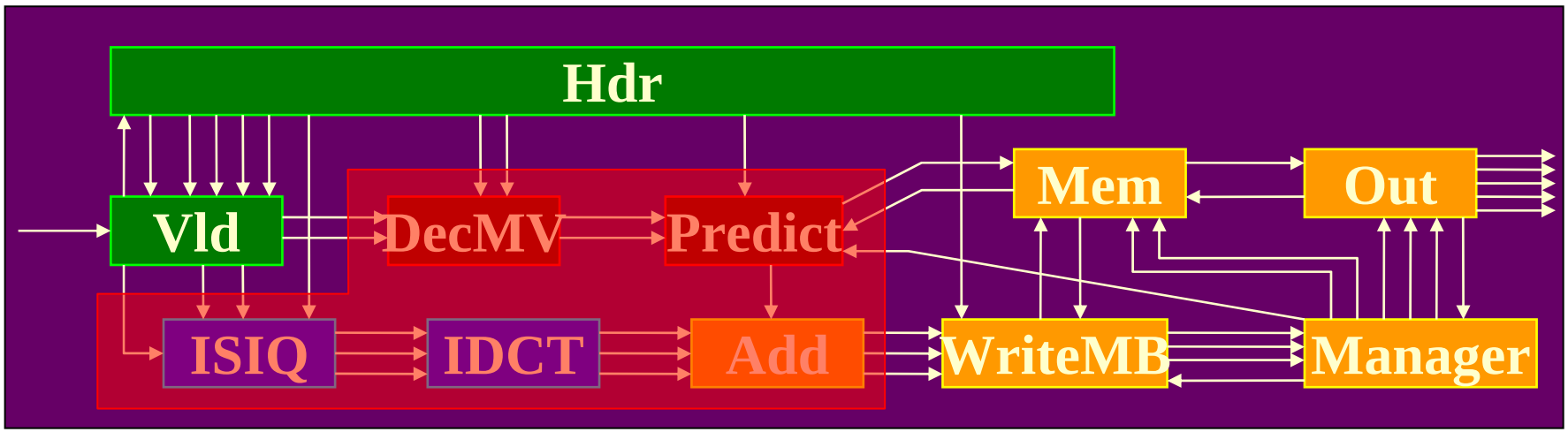
Finding a schedule on the Petri net



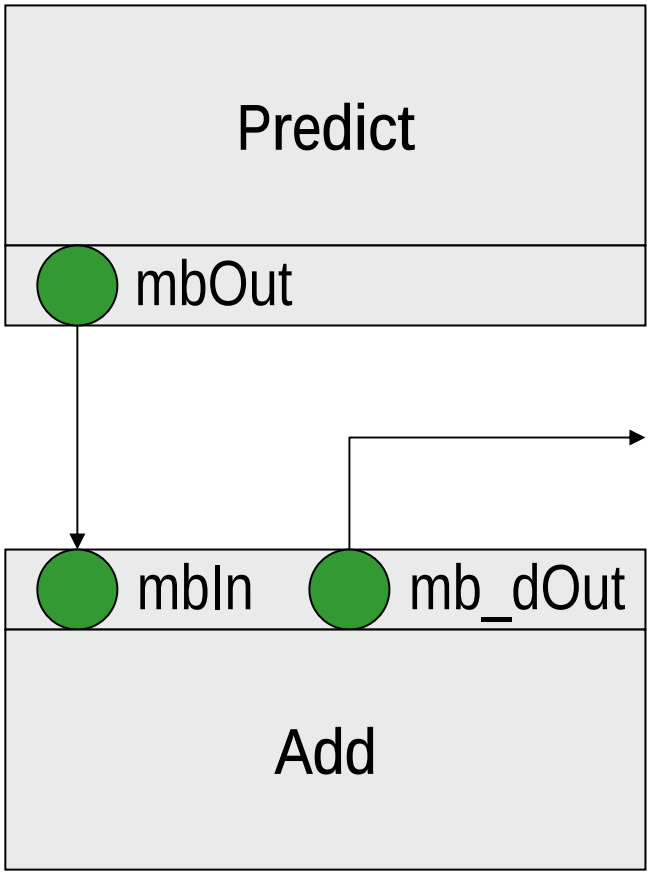


The MPEG2 decoder

- Part of a complete commercial video decoder
- Described in YAPI (developed at Philips)
  - based on Kahn Process Network
  - partially translated into FlowC to apply QSS
- 11 processes
  - 5000 lines of code of C++ code
- 45 FIFOs



The MPEG2 decoder




```
smbc = prop.skipped_cnt;
while (smbc > 0) {
    DoPredictionSkipped(<params>);
    Write(mbOut, mb_p);
    smbc--;
}
/* Other Predict stuff here */
```

```
smbc = mb_prop.skipped_cnt;
while (smbc > 0) {
    Read(mbIn, mb_p);
    Write(mb_dOut, mb_p);
    smbc--;
}
/* Other Add stuff here */
```

The MPEG2 decoder

```
smbc = prop.skipped_cnt;
while (smbc > 0) {
  DoPredictionSkipped(<params>);
  Write(mbOut, mb_p);
  smbc--;
}
/* Other Predict stuff here */
```

```
smbc = mb_prop.skipped_cnt;
while (smbc > 0) {
  Read(mbIn, mb_p);
  Write(mb_dOut, mb_p);
  smbc--;
}
/* Other Add stuff here */
```

- **Various optimizations still possible:**
  - remove unused variables
  - remove unused communications

```
Predict_smbc = Predict_prop.skipped_cnt;
Add_smbc = Add_ mb_prop.skipped_cnt
looplabel:
(void) (Add_smbc > 0);
if (Predict_smbc > 0) {
    DoPredictionSkipped(<params>);
    Write(mbOut, Predict_mb_p, 1);
    Read(mbIn, Add_mb_p, 1);
    Write(mb_dOut, Add_mb_p, 1);
    Predict_smbc--;
    Add_smbc--;
    goto looplabel;
} else if (!(Predict_smbc > 0) {
}
```

The MPEG2 decoder

	Total	MPEG			TestBench	OS
		Total	vld+hdr	5 blocks		
Philips	7.5	4.66	0.94	3.72	0.27	2.58
QSS	4.1	2.51	0.94	1.57	0.28	1.31

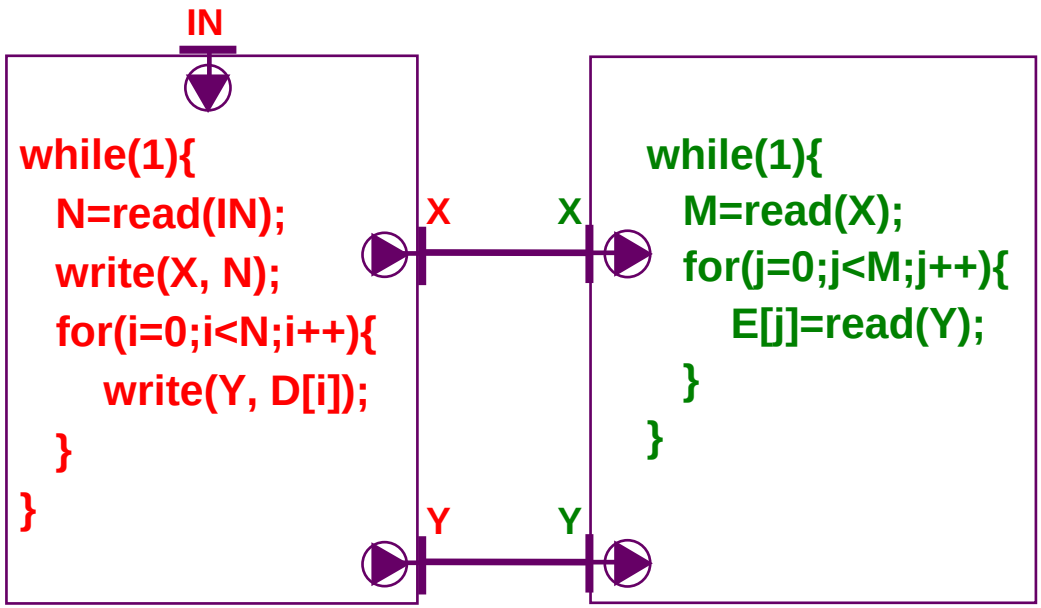
	5 blocks		
	Total	Computation	Communication
Philips	3.72	1.49	2.23
QSS	1.57	1.44	0.13

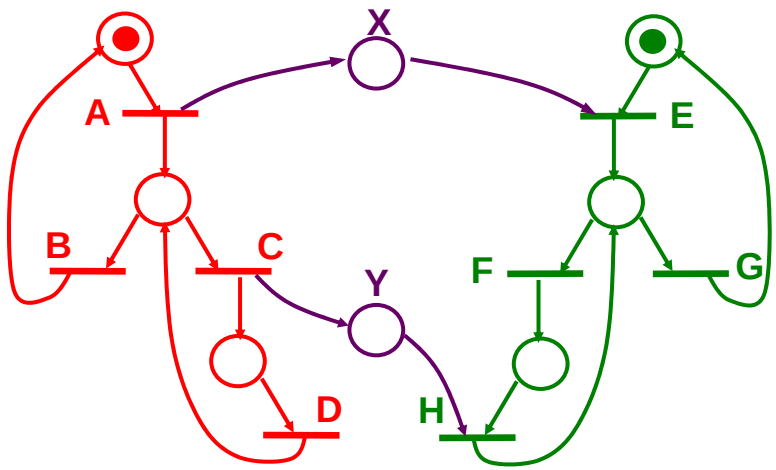
- Performance improved by 45%
  - reduction of communication (no internal FIFOs between statically scheduled processes)
  - reduction of run-time scheduling (OS)
  - no reduction in computation

Outline

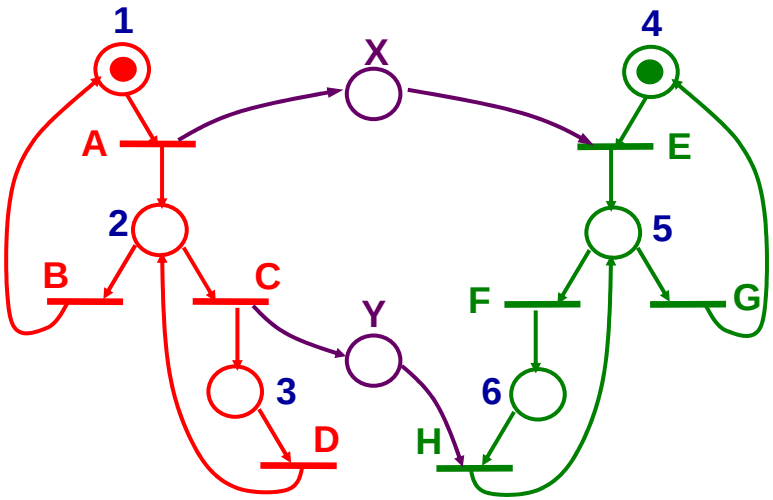
- The problem and an approach
  - Petri Net: a model for capturing the behavior of a program
  - Definition of a schedule
  - An algorithm
- An application to MPEG2 decoder
  - The effectiveness
  - The outstanding problem: False Path Problem
- Approaches for the False Path Problem
  - Conventional approaches
  - Cong's observation
  - Future directions

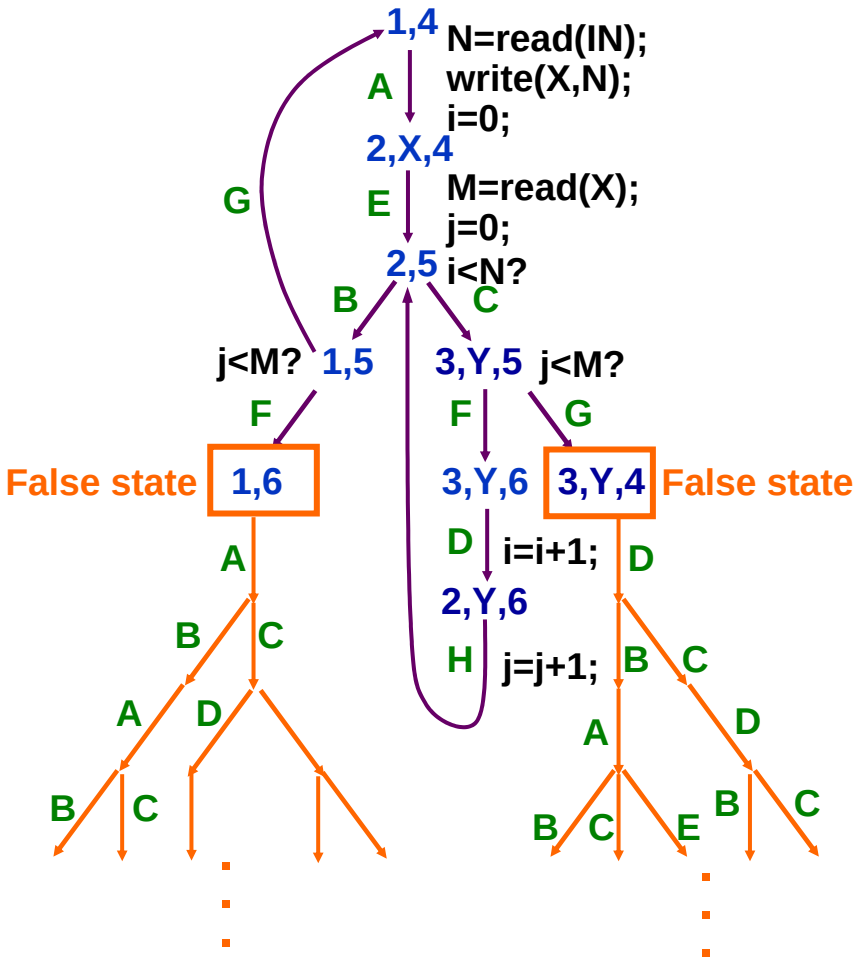
False paths in scheduling concurrent programs





False paths in scheduling concurrent programs

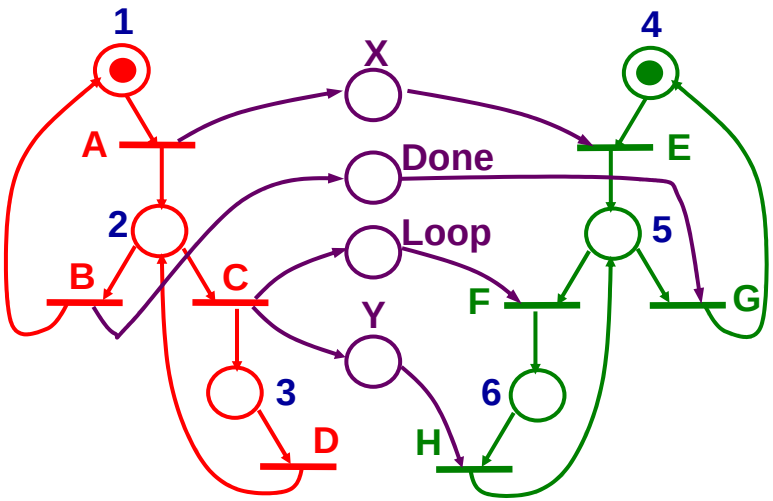


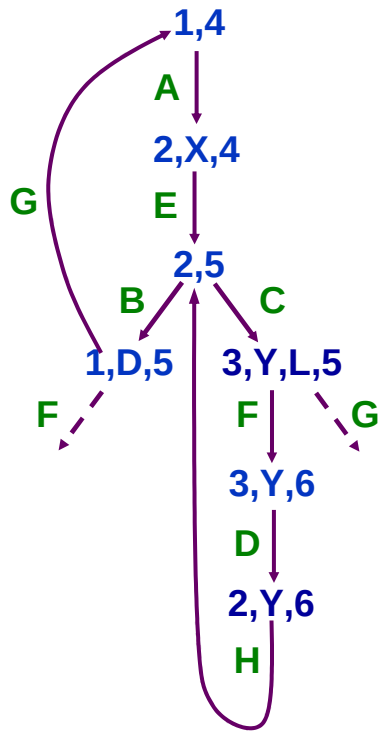


- The false path problem arises very often in practice.
- With this problem, the scheduling algorithm blows up, or produces huge schedules.

Our previous approach for the false path problem

1. **Manually change the input program to eliminate the false paths:**
   [Arrigoni et. al, 2002]



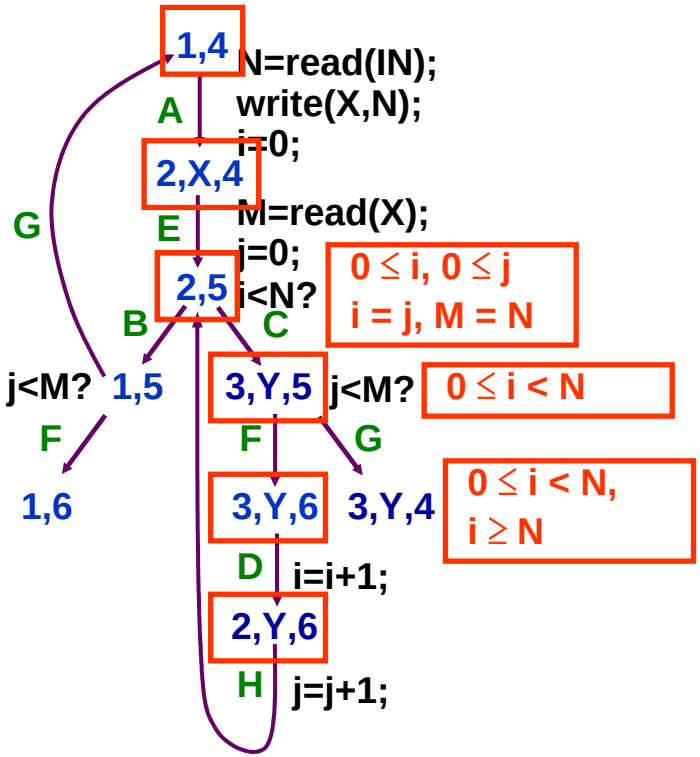


- Effective if specified correctly.
- The additional burden to the user may not be practical.

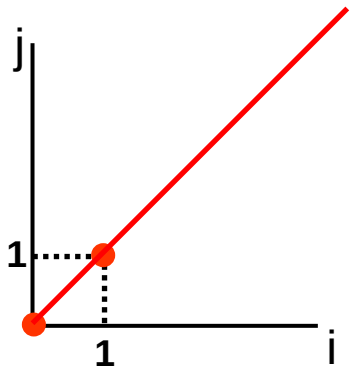
Our previous approach for the false path problem

2. Compute sets of values of variables at each state:

[Cousot, 1976]



The convex hull computation at {2,5}



- Restrictions on arithmetic operations to be handled.
- Restrictions on the problem sizes.

Outline

- The problem and an approach
  - Petri Net: a model for capturing the behavior of a program
  - Definition of a schedule
  - An algorithm

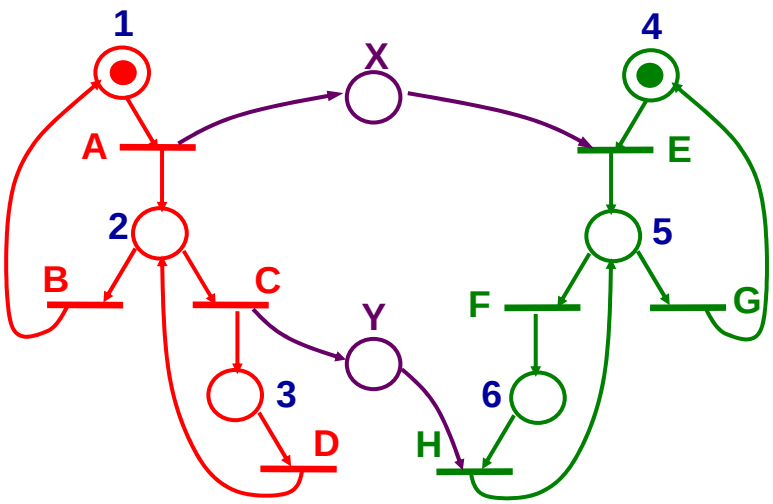
- An application to MPEG2 decoder
  - The effectiveness
  - The outstanding problem: False Path Problem

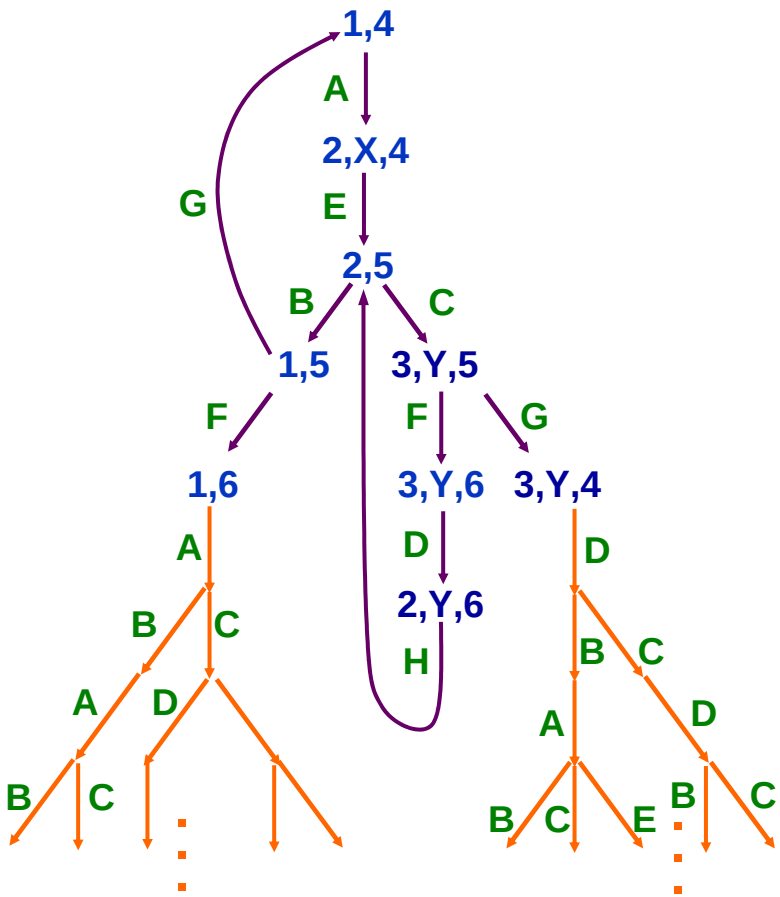
- Approaches for the False Path Problem
  - Conventional approaches
  - Cong's observation
  - Future directions

Cong's observation

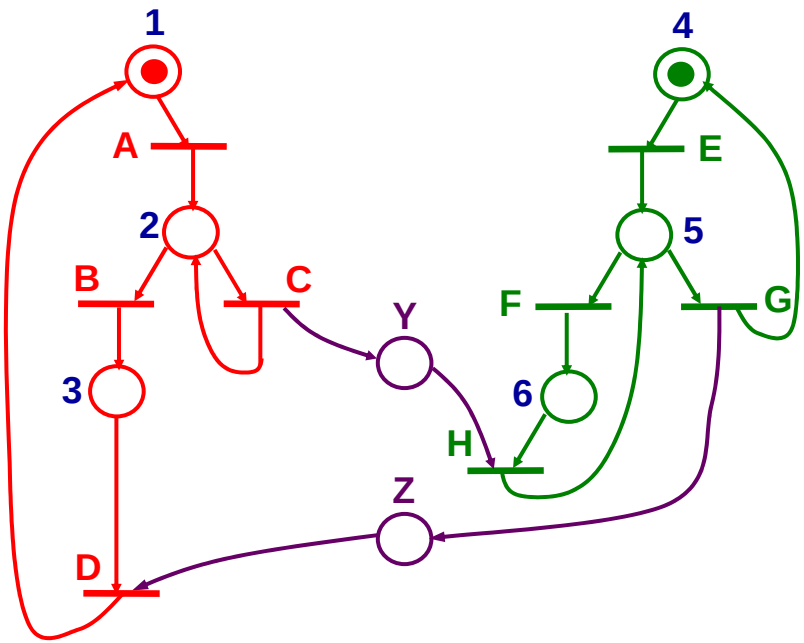
A certain pattern exists beyond the false states in the reachability tree:

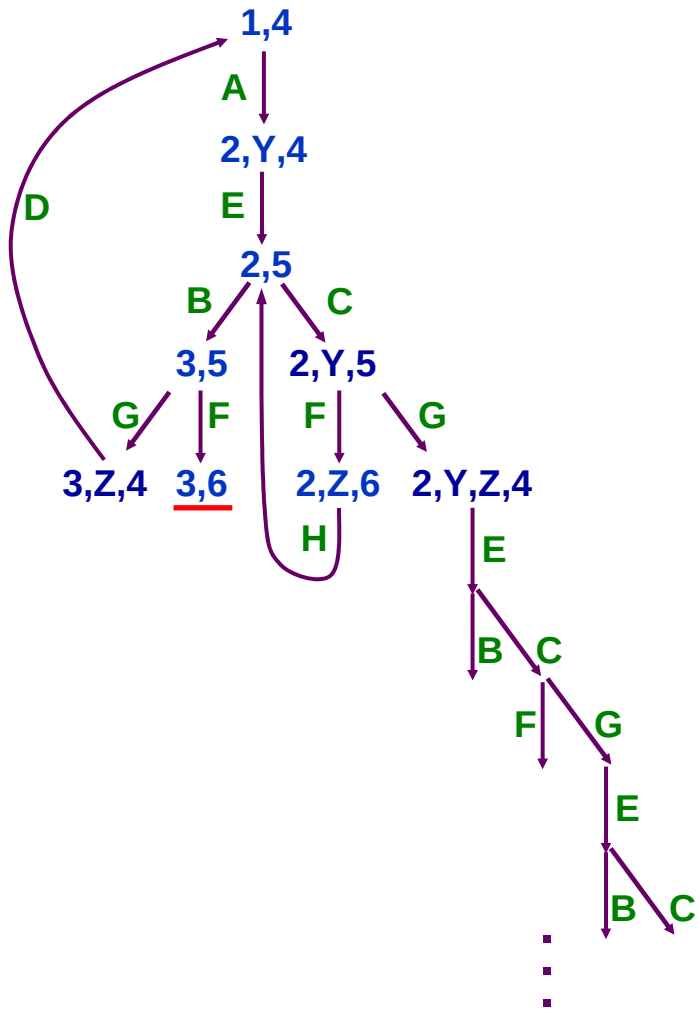
- The pattern is observed in typical dataflow applications we have in practice.
- The pattern makes a scheduler fail to find a (finite) schedule.
- The pattern is caused by a structural property of the input Petri net.





Cong's observation



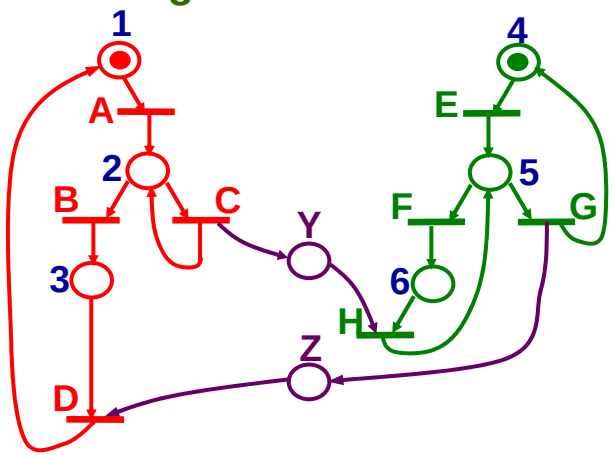


Cong's observation

- Transition dependency

  A transition *s* requires a transition *t*, $s \Rightarrow t$, if for any T-invariant of the Petri net, if it contains *s*, then it also contains *t*.

  – **T-invariant: a solution of the marking equations. I.e., a set of instances of transitions such that if all and only the instances of the set are fired, the resulting marking is same as before.**



T-invariants (minimal):

{A, B, D, E, G}

{C, F, H}

ECS1={B,C} and ECS2={F,G} are recurrent:

$B \Rightarrow G$ and $F \Rightarrow C$

$C \Rightarrow F$ and $G \Rightarrow B$

- Recurrent ECS's

  Two distinct ECS's, ECS1 and ECS2, are recurrent if there exist distinct elements $\{si, sj\} \subseteq$ ECS1 and $\{tk, tl\} \subseteq$ ECS2, such that $si \Rightarrow tk$ and $tl \Rightarrow sj$.

Cong's observation

Proposition

No schedule of a given Petri net contains recurrent ECS'es.

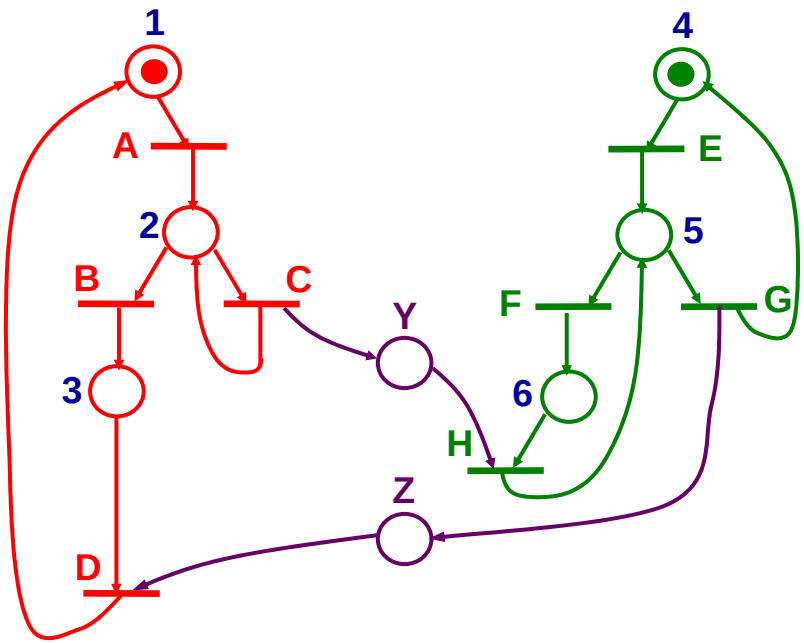
Proof:

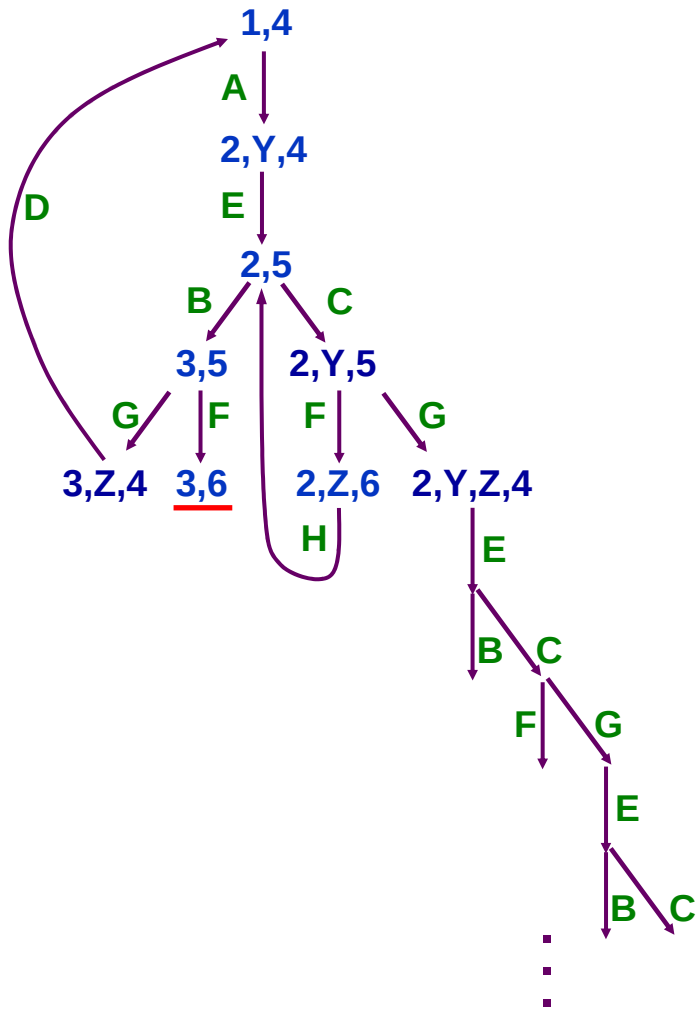
What to show: any attempt to include one of recurrent ECS'es in a schedule leads to either a deadlock or infinite paths.

Say ECS1 and ECS2 are recurrent, with {si, sj} $\subseteq$ ECS1 and {tk, tl} $\subseteq$ ECS2 and si $\Rightarrow$ tk and tl $\Rightarrow$ sj.

- A schedule needs to include all resolutions of an ECS contained in it.
- Any instance of ECS1 in the reachability tree has a path such that
  - the path starts with si, and
  - tk and sj do not appear in the path, and
  - the path ends with a deadlock, or else is infinite (no repeated marking in the path).
- Had a schedule contained ECS1, there is its instance for which the path above does not have any marking repeated in the pre-history of the path in the schedule.

Cong's observation





ECS1={B,C} and ECS2={F,G} are recurrent:

1. C $\Rightarrow$ F and G $\Rightarrow$ B
2. B $\Rightarrow$ G and F $\Rightarrow$ C

Cong's observation: summary

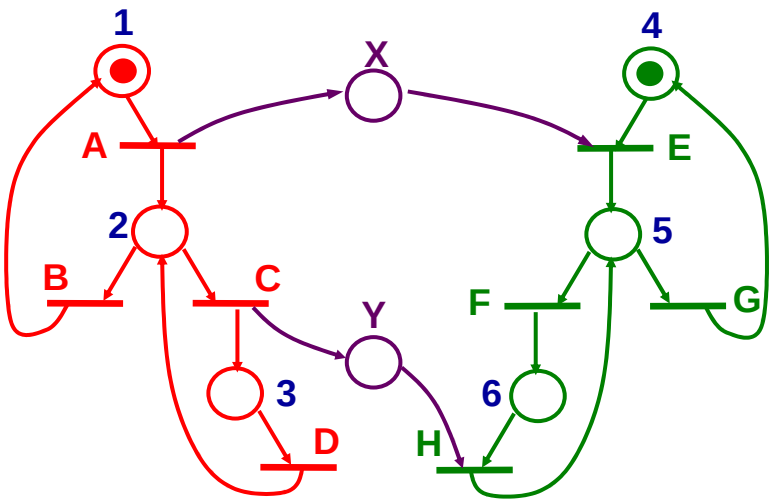
- Recurrent ECS'es lead the schedule search to either deadlock or endless.
- The recurrence between ECS'es is a structural property of a Petri net.
  - **The proposition holds regardless of markings.**
  - **The property can be found with a structural analysis.**

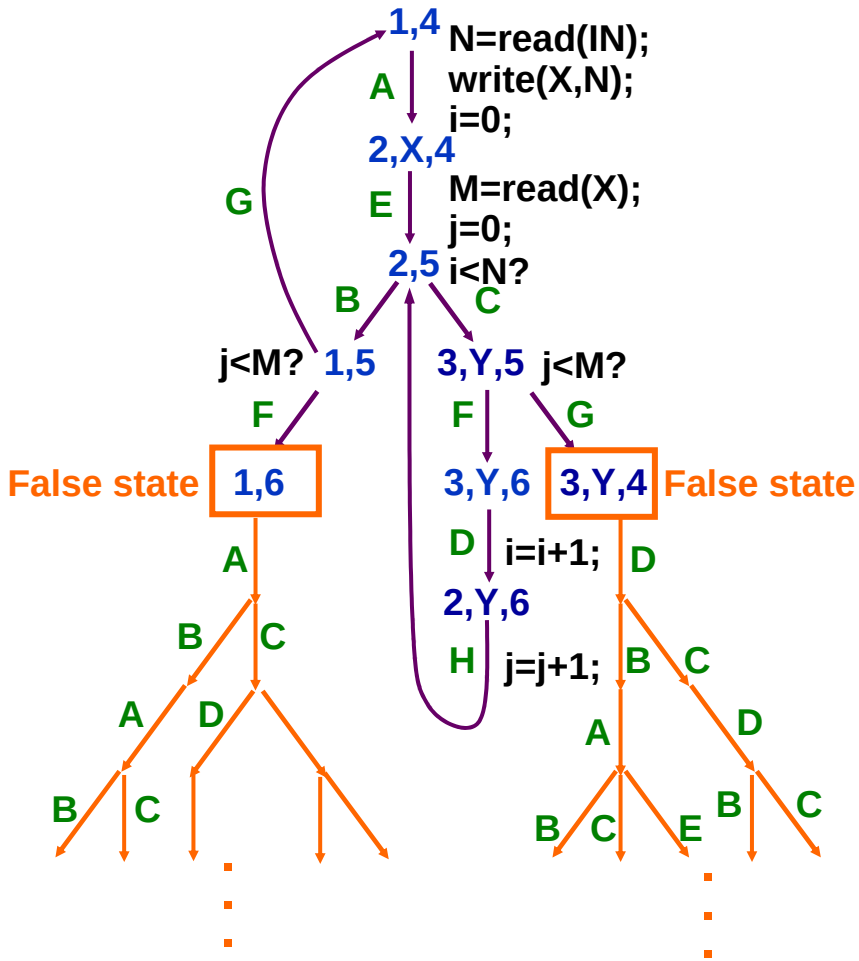
Directions of research with Cong's observation:

1. Application to the false path problem
2. Application to the schedulability analysis
3. Extension of the notion of recurrent ECS'es

Directions of research with Cong's observation

1. Application to the false path problem





Concluding remarks

- Static scheduling of concurrent programs finds attractive applications in practice.
- Petri nets with abstracted control flow seem to be a reasonable model, but the false path problem stands as a showstopper.
- Previous attempts directly looked at data-value analysis.
- Cong's observation deals with structural properties of the Petri nets:
  - A strong proposition, independent of markings
  - A mechanism of a failure of a schedule search revealed
  - While not directly applicable to the false path problem, it may lead to an effective way to address the problem.

Directions of research with Cong's observation

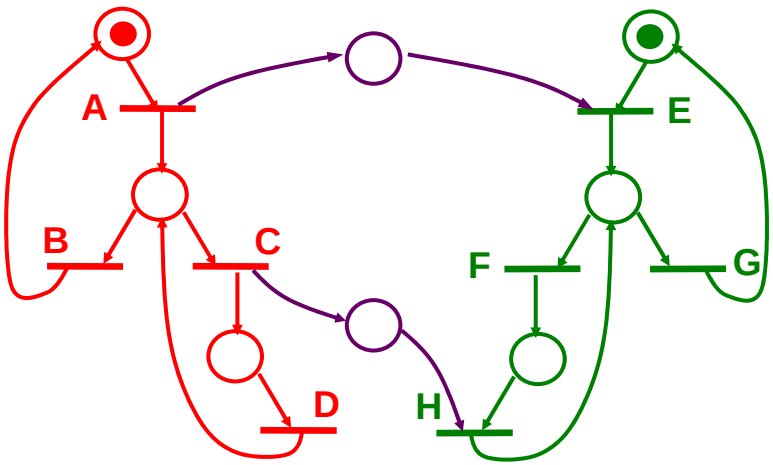
2. Application to the schedulability problem

Question: when does the other direction of Cong's proposition hold? I.e.

> Given two distinct ECS'es, suppose that no schedule contains either of them. Are these ECS'es recurrent then?

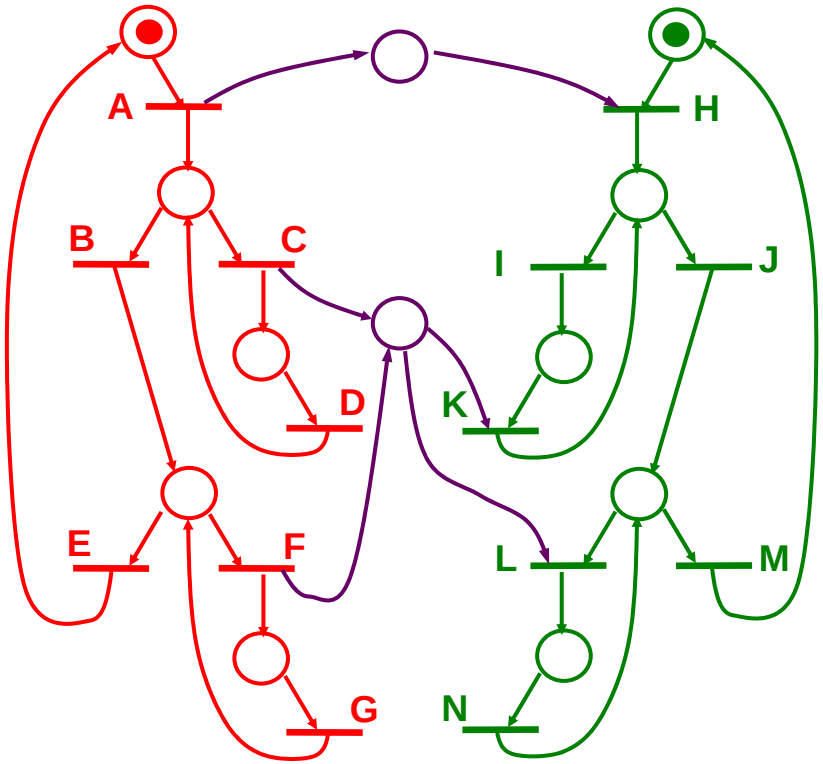
Directions of research with Cong's observation

3. Extension of the notion of recurrent ECS's



Directions of research with Cong's observation

3. Extension of the notion of recurrent ECS'es



ECS1={B,C} , ECS2={I,J} ,
ECS3={E,F} , ECS4={L,M}

$C \Rightarrow I \vee L$

$F \Rightarrow I \vee L$

$I \Rightarrow C \vee F$

$L \Rightarrow C \vee F$

$B \Leftrightarrow J$

$B \Leftrightarrow M$

$E \Leftrightarrow J$

$E \Leftrightarrow M$

None of them is recurrent to each other.